ADAM™ SmartBASIC™

FUJINET REFERENCE SECTION

(APPENDIX D)

INTRODUCING THE FUJINET COMMANDS

FujiNet is a network adapter that plugs into your ADAM's AdamNet bus and connects it to the Internet or to your own local network. This version of SmartBASIC adds seventeen new statements, all beginning with the letter N, that let your programs talk to it directly -- no PEEKs, POKEs or CALLs required.

The seventeen statements come in two families:

CONNECTION commands -- NOPEN, NCLOSE, NREAD, NWRITE, NSTATUS, NJSONPARSE, NJSONQUERY, NACCEPT, NLOGIN and NHTTPMODE -- move data over a network connection. Each takes a CHANNEL number (1 to 15) as its first argument, so you can hold several connections open at once. Most programs simply use channel 1.

FILE commands -- NCD, NDIR, NMKDIR, NRMDIR, NDEL, NLOAD and NSAVE -- work on files and directories stored on a network server. They take only a network path and always use channel 1.

Every N command may be typed in immediate mode or used inside a program, just like PRINT. (NLOAD and NSAVE are best used in immediate mode, like LOAD and SAVE.)

NETWORK PATHS

A network path names what you want to talk to. It always begins with N: followed by a protocol, a host, and (sometimes) a port and a path on that host:

```
"N:PROTOCOL://HOST[:PORT]/PATH"
```

The protocols available depend on your FujiNet's firmware; the common ones are HTTP, HTTPS, TCP, UDP, TELNET, TNFS and FTP. (THE NETWORK PROTOCOLS section, later in this appendix, catalogues every protocol and gives an example of each.) For instance:

```
"N:HTTP://FUJINET.ONLINE/HELLO.TXT"
"N:TCP://192.168.1.5:9000/"
"N:TNFS://TMA-3/GAMES/ADVENTURE.BAS"
```

THINGS TO REMEMBER

1. SmartBASIC strings hold at most 255 characters, so NREAD, NWRITE and NJSONQUERY move at most 255 bytes per call. To move more, loop -- see NREAD and the NFUJI program at the end of this appendix.
2. If no FujiNet is attached to your ADAM, any N command stops with ?ILLEGAL QUANTITY ERROR instead of hanging the computer.
3. When a command fails on the network -- a missing file, a refused connection, a wrong password --

it stops with ?Network Error and a code, like any other BASIC error. The NETWORK ERRORS section, later in this appendix, lists the codes and shows how to catch them with ONERR.

4. Always NCLOSE a channel when you are done with it.

QUICK REFERENCE

CONNECTION COMMANDS

```
NOPEN  c, path$, mode, trans   open a connection
NCLOSE c                       close it again
NREAD  c, var$, length         read up to 255 bytes
NWRITE c, var$, length         write up to 255 bytes
NSTATUS c, bw, conn, err       bytes waiting, connected,
                               error code
NJSONPARSE c                   parse reply as JSON
NJSONQUERY c, query$, var$     fetch one JSON value
NACCEPT c                      answer an incoming call
NLOGIN  c, user$, pass$        name & password for the
                               next NOPEN
NHTTPMODE c, n                 0=body 1=collect hdrs
                               2=get hdrs 3=set hdrs
                               4=post data
```

FILE COMMANDS (always channel 1)

```
NCD    path$                   set current net directory
NDIR   path$                   show a directory listing
NMKDIR path$                   make a directory
NRMDIR path$                   remove an empty directory
NDEL   path$                   delete a file
NLOAD  path$                   load an ASCII program
NSAVE  path$                   save program as ASCII
```

ARGUMENTS

```
c      channel number, 1 to 15
path$  network path: "N:PROTO://HOST[:PORT]/PATH"
proto  HTTP HTTPS TCP UDP TELNET SSH TNFS FTP
       SMB NFS SD GDRIVE CPM   (see THE NETWORK
       PROTOCOLS, later in this appendix)
mode   4=read  8=write  12=read/write  13=HTTP POST
trans  line-end translation:
       0=none  1=CR  2=LF  3=CR/LF
```

NOPEN

NOPEN opens a network connection on a channel. Its syntax is:

```
NOPEN <channel>, <path$>, <mode>, <trans>
```

channel = a number from 1 to 15. Every later command that names the same channel talks to this connection.

path$ = the network path to open, in quotes or in a string variable.

mode = how you intend to use the connection: 4 to read, 8 to write, 12 to read and write (use this for TCP conversations), 13 to make an HTTP POST.

trans = line-ending translation. Different computers end their text lines differently; the ADAM uses a carriage return (CR). Use 0 for no translation (binary data), 1 to translate network line endings to CR, 2 for LF, 3 for CR/LF.

Opening a channel also returns it to plain (protocol) mode if an earlier NJSONPARSE had switched it to JSON mode.

Test Program:

```
10 NOPEN 1,"N:HTTP://FUJINET.ONLINE/HELLO.TXT",4,0
20 NSTATUS 1,BW,CONN,ERR
30 PRINT BW;CONN;ERR
40 NCLOSE 1
```

Sample Run:

```
14 1 1
```

Fourteen bytes are waiting to be read, the connection is open, and the error code of 1 means "no error."

NCLOSE

NCLOSE closes the connection on a channel and lets the FujiNet reclaim it. Its syntax is:

```
NCLOSE <channel>
```

Close every channel you open, as soon as you are done with it. Closing a channel that is already closed is harmless.

Test Program:

```
10 NOPEN 1,"N:HTTP://FUJINET.ONLINE/HELLO.TXT",4,0
20 NCLOSE 1
```

Sample Run:

```
NONE
```

(The program opens a connection and closes it again, printing nothing.)

NREAD

NREAD reads incoming bytes from a channel into a string variable. Its syntax is:

```
NREAD <channel>, <var$>, <length>
```

length = the most bytes you want, 1 to 255. After the read, var$ holds exactly the bytes that arrived -- which may be fewer than you asked for. Use LEN(var$) to see how many came.

NREAD WAITS until the device has something to send. So that your program never waits forever, follow the golden rule of network reading: check NSTATUS first, and NREAD only when bytes are waiting. The loop below drains a whole document of any size, 255 bytes at a time; it is the pattern to copy into your own programs.

Test Program:

```
10 NOPEN 1,"N:HTTP://FUJINET.ONLINE/HELLO.TXT",4,0
20 NSTATUS 1,BW,CONN,ERR
30 IF BW=0 AND CONN=0 THEN 90
40 IF BW=0 THEN 20
50 L=BW : IF L>255 THEN L=255
60 NREAD 1,A$,L
70 PRINT A$;
80 GOTO 20
90 NCLOSE 1
```

Sample Run:

```
HELLO, ADAM!
```

NWRITE

NWRITE sends bytes from a string variable out over a channel. Its syntax is:

```
NWRITE <channel>, <var$>, <length>
```

length = how many characters of var$ to send, 1 to 255. If length is more than LEN(var$), only LEN(var$) characters are sent.

The channel must have been opened with a mode that allows writing: 8, 12 or 13.

Test Program:

```
10 NOPEN 1,"N:TCP://192.168.1.5:9000/",12,0
20 B$="HELLO FUJINET"
30 NWRITE 1,B$,LEN(B$)
40 NCLOSE 1
```

Sample Run:

```
NONE
```

(Nothing prints on the ADAM. The words HELLO FUJINET appear on the machine at address 192.168.1.5 that is listening on port 9000.)

NSTATUS

NSTATUS asks the FujiNet how a channel is doing and puts the answers into three numeric variables that you name. Its syntax is:

```
NSTATUS <channel>, <bw>, <conn>, <err>
```

bw = bytes waiting -- how many bytes you could NREAD right now. When more than 32767 bytes are waiting this number may show as negative; simply treat ANY value that is not 0 as "there is data to read."

conn = 1 while the connection is open, 0 once the other side has finished and disconnected.

err = the FujiNet's error code for the channel. 1 means no error, 136 means end of file; other values are protocol error codes.

When bw is 0 AND conn is 0, the transfer is complete: nothing is left to read and nothing more is coming. That test appears in every well-mannered receive loop (see NREAD).

NSTATUS itself never stops with a Network Error -- it is the tool for LOOKING at error codes. (The other commands stop when err is anything but 1 or 136; see NETWORK ERRORS, later in this appendix.)

Test Program:

```
10 NOPEN 1,"N:HTTP://FUJINET.ONLINE/HELLO.TXT",4,0
20 NSTATUS 1,BW,CONN,ERR
30 PRINT "WAITING:";BW
40 PRINT "CONNECTED:";CONN
50 PRINT "ERROR:";ERR
60 NCLOSE 1
```

Sample Run:

```
WAITING: 14
CONNECTED: 1
ERROR: 1
```

NJSONPARSE

Much of the information on today's Internet is served in a text format called JSON. NJSONPARSE tells the FujiNet to read the whole reply waiting on a channel and parse it as a JSON document. Its syntax is:

```
NJSONPARSE <channel>
```

Use it right after NOPEN, and follow it with NJSONQUERY to pick out the values you want -- the FujiNet does the hard work of understanding the document so your BASIC program doesn't have to.

NJSONPARSE switches the channel into JSON mode: after it, NREAD returns query results rather than the raw document text. To read a document as plain text again, open it afresh with NOPEN.

Test Program:

```
10 NOPEN 1,"N:HTTPS://FUJINET.ONLINE/FUJINET.JSON",4,0
20 NJSONPARSE 1
30 NJSONQUERY 1,"/0/content",C$
40 PRINT C$
50 NCLOSE 1
```

Sample Run:

```
(The CONTENT field of the document's first
record is printed.)
```

NJSONQUERY

After NJSONPARSE has parsed a document, NJSONQUERY fetches one value out of it into a string variable. Its syntax is:

```
NJSONQUERY <channel>, <query$>, <var$>
```

query$ = a path into the document, written as names and element numbers separated by slashes. Element numbers start at 0. Given the document

```
{ "news": [ { "title": "ADAM LIVES" },
            { "title": "FUJINET SHIPS" } ] }
```

the query "/news/0/title" fetches ADAM LIVES and "/news/1/title" fetches FUJINET SHIPS.

var$ = the string variable that receives the value (up to 255 characters of it).

Test Program:

```
10 NOPEN 1,"N:HTTPS://FUJINET.ONLINE/FUJINET.JSON",4,0
20 NJSONPARSE 1
30 NJSONQUERY 1,"/0/title",T$
40 NJSONQUERY 1,"/0/content",C$
50 PRINT T$ : PRINT C$
60 NCLOSE 1
```

Sample Run:

```
(The TITLE and CONTENT fields of the document's
first record are printed.)
```

NACCEPT

Every connection so far was a call your ADAM placed. NACCEPT is for the other direction: it answers a call another machine places to YOUR ADAM. Its syntax is:

```
NACCEPT <channel>
```

First open a TCP channel with no host -- just a port. The FujiNet then LISTENS on that port instead of calling out. When NSTATUS reports conn as 1, a caller is waiting; NACCEPT answers, and from then on NREAD and NWRITE converse with the caller. When the caller hangs up, the channel listens again -- another NACCEPT greets the next caller.

Test Program:

```
10 NOPEN 1,"N:TCP://:2000/",12,0
20 NSTATUS 1,BW,CONN,ERR
30 IF CONN=0 THEN 20
40 NACCEPT 1
50 B$="HELLO FROM ADAM!"
60 NWRITE 1,B$,LEN(B$)
70 NCLOSE 1
```

Sample Run:

```
NONE
```

(On another machine, connect with a command such as "nc ADAM-ADDRESS 2000" and be greeted by your ADAM.)

NLOGIN

Some servers want to know who you are. NLOGIN gives a channel the name and password that the NEXT NOPEN on that channel will sign in with. Its syntax is:

```
NLOGIN <channel>, <user$>, <pass$>
```

Use NLOGIN just BEFORE the NOPEN it applies to. The credentials stay with the channel until you change them; NLOGIN with two empty strings ("") forgets them.

SSH always requires a name and password. An SMB file share may ask for them. The anonymous protocols simply ignore them.

Test Program:

```
10 NLOGIN 1,"ADAM","SECRET"
20 NOPEN 1,"N:SSH://TMA-3/",12,1
30 NSTATUS 1,BW,CONN,ERR
40 PRINT "CONNECTED:";CONN
50 NCLOSE 1
```

Sample Run:

```
CONNECTED: 1
```

(The ADAM has signed in to the SSH server TMA-3 as user ADAM. What the remote computer prints next -- its greeting and prompt -- is read with the NREAD loop.)

NHTTPMODE

A web exchange is more than its body: there are also HEADERS, the labeled lines that describe a request and its reply. NHTTPMODE chooses which part of an HTTP channel NREAD and NWRITE act on. Its syntax is:

```
NHTTPMODE <channel>, <mode>

mode 0  the body (the normal state)
     1  collect: NWRITE names a reply
        header you want kept
     2  get: NREAD returns the kept reply
        headers, one per read
     3  set: NWRITE adds a header to the
        request, as "NAME: VALUE"
     4  post data: NWRITE supplies the
        body of a mode-13 POST
```

Set your headers before the first NREAD or NSTATUS -- the FujiNet performs the request the first time you ask about the reply. Return to mode 0 to read the body.

Test Program:

```
10 NOPEN 1,"N:HTTP://FUJINET.ONLINE/HELLO.TXT",4,0
20 NHTTPMODE 1,3
30 H$="X-CLIENT: ADAM"
40 NWRITE 1,H$,LEN(H$)
50 NHTTPMODE 1,0
60 NSTATUS 1,BW,CONN,ERR
70 PRINT BW;CONN;ERR
80 NCLOSE 1
```

Sample Run:

```
14 1 1
```

(The document is fetched as usual, but the request carried the extra header X-CLIENT: ADAM.)

NCD

NCD sets your current network directory, just as walking into a room saves you from giving your full street address every time. Its syntax is:

```
NCD <path$>
```

Later network paths that do not name a protocol and host are taken relative to this prefix.

Test Program:

```
10 NCD "N:TNFS://TMA-3/GAMES"
```

Sample Run:

```
NONE
```

(TMA-3 here and on the following pages is the name of a TNFS file server on the local network. After this command, network files can be named relative to the GAMES directory on that server.)

NDIR

NDIR prints a directory listing from a network file server on your screen -- it is CATALOG for the network. Its syntax is:

```
NDIR <path$>
```

Test Program:

```
10 NDIR "N:TNFS://TMA-3"
```

Sample Run:

```
GAMES/
UTILS/
README.TXT
NFUJI.BAS
```

NMKDIR

NMKDIR makes a new directory on a network file server. Its syntax is:

```
NMKDIR <path$>
```

The server must allow you to write there; a protocol like HTTP that has no directories will refuse.

Test Program:

```
10 NMKDIR "N:TNFS://TMA-3/NEWDIR"
```

Sample Run:

```
NONE
```

(A directory named NEWDIR now exists on the server. Prove it with NDIR "N:TNFS:

NRMDIR

NRMDIR removes a directory from a network file server. Its syntax is:

```
NRMDIR <path$>
```

The directory must be empty -- delete its files first with NDEL.

Test Program:

```
10 NMKDIR "N:TNFS://TMA-3/NEWDIR"
20 NRMDIR "N:TNFS://TMA-3/NEWDIR"
```

Sample Run:

```
NONE
```

(The directory is created and then removed again.)

NDEL

NDEL deletes a file from a network file server. Its syntax is:

```
NDEL <path$>
```

Be careful: like DELETE on a data pack, NDEL does not ask "are you sure?"

Test Program:

```
10 NDEL "N:TNFS://TMA-3/OLDFILE.TXT"
```

Sample Run:

```
NONE
```

(OLDFILE.TXT is gone from the server.)

NLOAD

NLOAD fetches a SmartBASIC program stored as a plain text (ASCII) listing on a network server and enters it, line by line, exactly as if you were typing it. Its syntax is:

```
NLOAD <path$>
```

Because the lines are entered like typed lines, they MERGE with whatever program is already in memory. For a clean load, type NEW first.

Use NLOAD in immediate mode, like LOAD -- not from inside a running program, because it rewrites program memory as it goes.

Test Program:

```
NEW
NLOAD "N:TNFS://TMA-3/NFUJI.BAS"
LIST
```

Sample Run:

```
10 REM ***********************************
20 REM * FUJINET DEMO FOR SMARTBASIC 1.X *
...
```

(The NFUJI program has been fetched from the server and is ready to RUN.)

NSAVE

NSAVE writes the program in memory to a network server as a plain text (ASCII) listing -- the same form LIST shows you. Its syntax is:

```
NSAVE <path$>
```

A program saved with NSAVE is loaded back with NLOAD. Because the file is ordinary text, it can also be read, printed or edited on any modern computer -- NSAVE and NLOAD together are a bridge between your ADAM and the rest of the world.

Use NSAVE in immediate mode, like SAVE.

Test Program:

```
NSAVE "N:TNFS://TMA-3/MYPROG.BAS"
```

Sample Run:

```
NONE
```

(The program in memory is now stored on the server as MYPROG.BAS.)

The word after N: in a network path chooses the PROTOCOL -- the language the FujiNet will speak on your behalf. The firmware in your FujiNet knows thirteen. Any of them may be opened with NOPEN; the ones tagged "files" also store files, and so work with the file commands NCD, NDIR, NMKDIR, NRMDIR, NDEL, NLOAD and NSAVE as well.

HTTP AND HTTPS

```
"N:HTTP://HOST[:PORT]/DOCUMENT"
```

The World Wide Web. Open with mode 4 and read: the document arrives. Open with mode 13 to POST: NWRITE your data, then NREAD the server's reply. HTTPS is HTTP with secrecy, and the FujiNet does all the secret-keeping -- certificates and encryption never trouble your BASIC program. NLOAD can fetch a program straight off the web, too.

```
NOPEN 1,"N:HTTPS://FUJINET.ONLINE/HELLO.TXT",4,0
```

TCP

```
"N:TCP://HOST:PORT/"
```

A plain two-way socket -- the bare metal of nearly everything on the Internet. Open with mode 12 and converse freely with NREAD and NWRITE; NSTATUS tells you when the other side hangs up. If the port is left out, 23 is assumed. Give no host at all ("N:TCP://:2000/") and the FujiNet LISTENS on that port instead: your ADAM can take calls as well as place them (see NACCEPT).

```
NOPEN 1,"N:TCP://192.168.1.5:9000/",12,0
```

UDP

```
"N:UDP://HOST:PORT/"
```

Postcards rather than a telephone call: each NWRITE sends one datagram and each NREAD collects one, with no connection and no promise of delivery or order. Fast and simple -- good for game and sensor traffic on your own network. Leave HOST empty ("N:UDP://:5000/") to listen for datagrams on a port instead.

```
NOPEN 1,"N:UDP://192.168.1.5:5000/",12,0
```

TELNET

```
"N:TELNET://HOST[:PORT]/"
```

For conversing with the bulletin boards (BBSes) that still flourish on the Internet. TELNET is TCP plus etiquette: the far end may ask questions about your terminal, and the FujiNet answers them for you. The port is 23 unless you say otherwise.

```
NOPEN 1,"N:TELNET://BBS.FOZZTEXX.COM/",12,1
```

SSH

```
"N:SSH://HOST[:PORT]/"
```

The secure remote terminal of the Unix world. Give the channel your name and password with NLOGIN first, then open with mode 12: what you NWRITE arrives at the remote computer as keystrokes, and what it prints comes back through NREAD -- encrypted both ways, with the FujiNet doing all the mathematics.

```
10 NLOGIN 1,"ADAM","SECRET"
20 NOPEN 1,"N:SSH://TMA-3/",12,1
```

TNFS (files)

```
"N:TNFS://HOST/PATH"
```

The FujiNet community's own file server, and the natural home of the file commands. Free server programs (look for "tnfsd") run on Windows, Macintosh and Linux, so any spare computer can serve your ADAM. The TMA-3 server in this appendix's examples speaks TNFS.

```
NDIR "N:TNFS://TMA-3"
```

FTP (files)

```
"N:FTP://HOST/PATH"
```

The Internet's elder statesman of file moving. The FujiNet signs you in as an "anonymous" guest -- just what the Internet's public FTP archives expect. Read files, list directories, and write where a server permits it.

```
NDIR "N:FTP://FTP.GNU.ORG/"
```

SMB (files)

"N:SMB://SERVER/SHARE/PATH"

File sharing the Windows way -- spoken by Windows machines, by NAS boxes, and by Macintosh and Linux computers running Samba. If the share asks for a name and password, write them into the path:

```
NDIR "N:SMB://NAME:SECRET@SERVER/PUBLIC"
```

NFS (files)

"N:NFS://HOST/EXPORT/PATH"

File sharing the Unix way. Name the host, the exported directory, and the path within it.

```
NLOAD "N:NFS://TMA-3/EXPORT/HELLO.BAS"
```

SD (files)

"N:SD:///PATH"

The memory card plugged into the FujiNet itself. The three slashes are deliberate -- there is no host to name, because this "server" is inches from your ADAM. A handy waystation: NSAVE a program to the card today, NLOAD it anywhere tomorrow.

```
NLOAD "N:SD:///GAMES/HELLO.BAS"
```

GDRIVE (files)

"N:GDRIVE:///PATH"

A disk drive in the clouds: your Google Drive. Introduce your FujiNet to your Google account once, on the FujiNet's configuration web page; from then on the Drive reads and writes like any file server. (Three slashes again -- the host is understood.)

```
NDIR "N:GDRIVE:///"
```

CPM

"N:CPM:///"

Not a network at all. Opening this channel switches on a complete CP/M 2.2 computer living inside the FujiNet. NWRITE sends its keyboard, NREAD collects its screen; its disks live on the FujiNet's memory card. The receive loop from the NREAD page, plus a GET for your keystrokes, turns the ADAM into a CP/M terminal.

```
NOPEN 1,"N:CPM:///",12,0
```

For completeness: the firmware also answers to TEST, which manufactures sample data for checking the

FujiNet itself, and to SSH.KEYGEN and SSH.COPYID, two helpers that make and install the keys used for password-less SSH logins.

Things go wrong on networks: servers vanish, files go missing, passwords are mistyped. When a FujiNet command fails, SmartBASIC treats it like any other BASIC error. The command stops, and the screen shows

```
?Network Error 170
```

with the line number added when a program is running. The number is the FujiNet's own code for what went wrong:

```
  1  no error
132  command not understood here
136  end of file (expected; never raised)
138  timed out
144  general failure
165  bad network path
167  access denied
170  file not found
200  connection refused
201  network unreachable
204  connection reset by the other side
207  not connected
209  no caller waiting (NACCEPT)
212  wrong name or password
213  the reply is not JSON (NJSONPARSE)
```

Three commands never stop this way: NSTATUS (it is how you LOOK at error codes), NCLOSE and NLOGIN. And one subtlety: the FujiNet does not fetch a web document until you first read or ask about it, so a bad HTTP address may sail through NOPEN and stop the first NREAD instead.

A network error ends your program unless you catch it with ONERR GOTO, just as with any other error. In the handler, ERRNUM returns 37 -- the BASIC error number shared by all network errors -- and NSTATUS fetches the FujiNet code. After an error the channel may still be open; NCLOSE it before you retry.

Test Program:

```
10 ONERR GOTO 900
20 NDEL "N:TNFS://TMA-3/NOFILE.TXT"
30 PRINT "DELETED" : END
900 IF ERRNUM<>37 THEN STOP
910 NSTATUS 1,BW,CONN,ERR
920 PRINT "NETWORK TROUBLE, CODE";ERR
930 NCLOSE 1
```

Sample Run:

```
NETWORK TROUBLE, CODE 170
```

(There is no NOFILE.TXT on the server, so NDEL fails with code 170, file not found. The handler at 900 catches it instead of letting the program stop.)

The program NFUJI.BAS, supplied with this version of SmartBASIC, exercises the whole FujiNet command set in four short, independent parts. RUN executes part one; the other parts are started from immediate mode with GOTO 1000, GOTO 2000 and GOTO 3000. Each part is listed and explained below.

PART ONE: FETCH A DOCUMENT FROM THE WEB (RUN)

```
100 REM --- HTTP GET, PRINT TO SCREEN ---
110 NOPEN 1,"N:HTTP://WWW.GNU.ORG/LICENSES/GPL-3.0.
    TXT",4,0
120 NSTATUS 1,BW,CONN,ERR
130 IF BW=0 AND CONN=0 THEN 200
140 IF BW=0 THEN 120
150 L=BW : IF L>255 THEN L=255
160 NREAD 1,A$,L
170 PRINT A$;
180 GOTO 120
200 NCLOSE 1
210 PRINT : PRINT "--- DONE ---"
220 END
```

(Line 110 is a single program line; it is shown folded here to fit the page.)

Line 110 opens channel 1 on a document at the GNU project's web site, mode 4 (read only), translation 0.

Lines 120–180 are the receive loop from the NREAD page, the beating heart of every download. Line 120 asks how things stand. Line 130 spots the finish: nothing waiting AND disconnected means the document is done. Line 140 goes back to ask again if the connection is alive but no bytes have arrived yet. Line 150 clamps the chunk to 255, the most a string can carry. Line 160 reads the chunk; line 170 prints it with a trailing semicolon so PRINT adds no extra line endings of its own -- the carriage returns inside the document provide the line breaks. Line 180 loops for the next chunk.

Lines 200–220 close the channel and sign off.

PART TWO: READ A JSON FEED (GOTO 1000)

```
1010 REM JSON EXAMPLE (RUN: GOTO 1000)
1020 NOPEN 1,"N:HTTPS://FUJINET.ONLINE/FUJINET.
     JSON",4,0
1030 NJSONPARSE 1
1040 NJSONQUERY 1,"/0/content",C$
1050 PRINT C$
1060 NCLOSE 1
1070 END
```

(Line 1020 is likewise one program line, folded here.)

Line 1020 opens the FujiNet news feed -- note HTTPS: the FujiNet handles the secure connection by itself. Line 1030 has the FujiNet parse the reply as JSON. Line 1040 asks for one value: element 0 of the

document (its first record), then that record's CONTENT field. The text lands in C$ and line 1050 prints it. No receive loop is needed -- the FujiNet holds the parsed document and serves up just the value asked for.

PART THREE: A NETWORK FILE SERVER (GOTO 2000)

```
2010 REM TNFS FILESYSTEM EXAMPLE
2020 NCD "N:TNFS://TMA-3"
2030 NDIR "N:TNFS://TMA-3"
2040 NMKDIR "N:TNFS://TMA-3/NEWDIR"
2050 NRMDIR "N:TNFS://TMA-3/NEWDIR"
2060 END
```

TMA-3 is the name of a TNFS file server on the local network; substitute your own server's name or address. Line 2020 makes it the current network directory, line 2030 prints its catalog on the screen, and lines 2040-2050 make a directory and remove it again -- proof of a round trip.

PART FOUR: TALK TO ANOTHER COMPUTER (GOTO 3000)

```
3010 REM WRITE/POST EXAMPLE
3020 NOPEN 1,"N:TCP://192.168.1.5:9000/",12,0
3030 B$="HELLO FUJINET"
3040 NWRITE 1,B$,LEN(B$)
3050 NCLOSE 1
3060 END
```

Line 3020 opens a raw TCP conversation with the machine at 192.168.1.5, port 9000 -- mode 12, because a conversation needs both reading and writing. Line 3040 sends the thirteen characters of B$. On the other machine, listen with a command such as "nc -l 9000" and watch the ADAM say hello.

FOR THE CURIOUS: BEHIND THE SCENES

You can use every command in this appendix without reading this page. For the technically inclined, here is what happens underneath.

Each channel is an EOS character device on the AdamNet bus: channel 1 is device 9, channel 2 is device 10, and so on up to channel 15, device 23. The N commands drive these devices through EOS itself -- the same operating system calls that run your printer and data packs -- so they coexist peacefully with everything else on the bus.

Every command is a small packet whose first byte says what to do (O to open, C to close, S for status, W to write, and so on), written to the device; replies come back as device reads. AdamNet delivers up to 1024 bytes per read, so SmartBASIC keeps an internal buffer and hands your program up to 255 bytes at a time from it. NSTATUS counts those buffered bytes in with the bytes still on the FujiNet, so the receive loop's arithmetic always comes out right.

AdamNet transfers are retried automatically on timeout, and before any command is attempted, SmartBASIC checks that the device actually exists on the bus -- which is how a missing FujiNet earns a polite ?ILLEGAL QUANTITY ERROR instead of a frozen computer.

The FujiNet itself does the heavy lifting: domain names, TCP, the secure side of HTTPS, JSON parsing -- work far beyond a 3.58 MHz Z80 -- all happen on the adapter, and your ADAM simply reaps the results.

* * *

This appendix documents the FujiNet extension to SmartBASIC 1.x and is a community supplement to the ADAM™ SmartBASIC™ Programming Manual, Revised Edition (Coleco Industries, 1984).